CS 341: ALGORITHMS

Lecture 14: graph algorithms V = single source shortest path
Readings: see website

Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

PROBLEM: SINGLE SOURCE SHORTEST PATHS (SSSP)

- Input: graph $G = (V,E)$ and a non-negative weight function $w(e)$ defined for every edge $e$
- Problem: for every node $u \neq v$, output a path $u \rightarrow v$ with the smallest total weight (among all paths $u \rightarrow v$)
- I.e., each path $P$ should minimize $w(P) = \sum_{e \in P} w(e)$

And so on... one path for each node.

APPLICATION: DRIVING DISTANCE TO MANY POSSIBLE DESTINATIONS

- Single source: from where you are
- Shortest path to all destinations
- Display a subset of destinations
- Include the optimal distances computed using SSSP algorithm
- Other heuristics... traffic? Lights?
- Weights can combine many factors

Otherwise use BFS to find shortest sequence of waypoints with Trevor Brown!

Dijkstra's Algorithm

ILLUSTRATIVE EXAMPLE
**PROOF**

- **Theorem**: At the end of the algorithm, for all $u$, \( \text{dist}(u) \) is exactly the total weight of the shortest \( s \to u \) path.
- We prove this in two parts:
  - \( \text{dist}(u) \leq \) the total weight of the shortest \( s \to u \) path
  - \( \text{dist}(u) \geq \) the total weight of the shortest \( s \to u \) path

**DISTANCE SHORTEST PATH**

- Let \( P \) be any arbitrary \( s \to u \) path \( v_0 \to v_1 \to \cdots \to v_k \) where \( v_0 = s \) and \( v_k = u \)
- For any index \( i \), let \( L_i \) denote \( w(v_i \to v_{i+1}) \)
- We prove by induction: \( \text{dist}(v_i) \leq L_j \) for all \( j \)

**DISTANCE SHORTEST PATH**

- Let \( P' \) be the path \( s \to \cdots \to \text{pred}(\text{pred}(u)) \to \text{pred}(u) \to u \)
  - i.e., the reverse of following \( \text{pred} \) pointers from \( u \) back to \( s \)
- We show \( \text{dist}(u) \) is as long as this path (and hence as long as the shortest path)
- Denote the nodes in \( P' \) by \( v_0, v_1, \ldots, v_k \) where \( v_0 = s \) and \( v_k = u \)
- Let \( L_j = w(v_0 \to v_1 \to \cdots \to v_j) \)
- Prove by induction: \( \forall j: \text{dist}(v_j) = L_j \)
- Base case: \( \text{dist}(v_0) = \text{dist}(s) = 0 = L_0 \)
Recall:

So \( \text{dist}\[u\] \) is both \( \leq \) and \( \geq \) to the length of the shortest \( s \rightarrow u \) path!
That means it’s equal to the length of the shortest path!

\[
\text{dist}[u] \geq \text{shortest } s \rightarrow u \text{ path}
\]

\[
P = v_0 \rightarrow \cdots \rightarrow v_k = s \rightarrow \cdots \rightarrow \text{pred[}\text{pred}[u]\text{]} \rightarrow \text{pred}[u] \rightarrow u
\]

\[
l_j = w(v_j, v_{j+1})
\]

\[
\text{Inductive step: suppose } v_{k+1} \text{ } \text{dist}[v_{k+1}] = l_{k+1}
\]

When we set \( \text{pred}[v_i] = v_{i+1} \), we set \( \text{dist}[v_i] = \text{dist}[v_{i+1}] + w(v_{i+1}, v_i) \)

\[
\text{if } \text{dist}[u] + w(u, v) < \text{dist}[v] \text{ } \text{dist}[v] = \text{dist}[u] + w(u, v)
\]

\[
\text{pred}[v] = u
\]

By I.H., \( \text{dist}[v_j] = l_{j+1} + w(v_{j+1}, v_j) \)

By definition \( l_j = l_{j+1} + w(v_{j+1}, v_j) \)

So \( \text{dist}[v_j] = \text{length of shortest path } v_j \rightarrow u \)

So \( l_j = \text{both } \geq \) and \( \leq \) to the length of the shortest \( s \rightarrow u \text{ path} \)

So \( \text{dist}[u] = l_j \)

Outputting actual shortest path(s)?

To compute the actual shortest path \( s \rightarrow t \)

- inspect \( \text{pred}[t] \)
  - If it is NULL, there is no such path
  - Otherwise, follow \( \text{pred} \) pointers back to \( s \), and return the reverse of that path

Website demonstrating Dijkstra’s alg


Bellman-Ford

- Single-source shortest path in a graph with possibly negative edge weights, but no negative cycles
- \( O(nm) \)
Shortest Paths and Negative Weight Cycles

Subsequent algorithms we will be studying will solve shortest path problems as long as there are no cycles having negative weight.

If there is a negative weight cycle, then there is no shortest path (why?).

There is still a shortest simple path, but there are apparently no known efficient algorithms to find the shortest simple paths in a graph containing negative weight cycles.

If there are no negative weight cycles, we can assume WLOG that shortest paths are simple paths (any path can be replaced by a simple path having the same weight).

Negative weight edges in an undirected graph are not allowed, as they would give rise to a negative weight cycle (consisting of two edges) in the associated directed graph.

Edges happen to be processed left to right by the inner loop.

• It technically suffices to do one iteration of the outer loop.

• After 1 iterations of the outer for-loop:
  - If \( D[u] = \infty \), it is equal to the weight of some path \( s \rightarrow u \).
  - If there is a path \( P = \langle s \rightarrow u \rangle \) with at most \( k \) edges, then \( D[u] \leq w(P) \).
  - So, after \( n \) iterations, if \( P \) with at most \( n \) edges, then \( D[u] \leq w(P) \).
  - Note: any more edges would create a cycle.

• So, if \( u \) is reachable from \( s \), then \( D[u] \) is the length of the shortest simple path (no cycles) from \( s \) to \( u \).

Of course every simple path has at most \( n - 1 \) edges.

So what if we do another iteration and some \( D[u] \) improves?

There is a negative cycle!

Best Case Execution

If technically suffices to do one iteration of the outer loop.

Worst Case Execution

Since the longest possible path without a cycle can be \( n - 1 \) edges, the edges must be scanned \( n - 1 \) times to ensure the shortest path has been found for all nodes.

Dijkstra’s is similar, but consistently achieves good ordering using its priority queue.

Why Bellman-Ford Works

• Not going to prove this (by induction), but the crucial lemma is:

• After \( i \) iterations of the outer for-loop:
  - If \( D[u] = \infty \), it is equal to the weight of some path \( s \rightarrow u \).

• After \( n \) iterations of the outer for-loop, if \( D[u] = \infty \), then there is a path \( P \) with at most \( n \) edges, then \( D[u] \leq w(P) \).

• So, after \( n \) iterations, if \( P \) with at most \( n \) edges, then \( D[u] \leq w(P) \).

• Note: any more edges would create a cycle.

• So, if \( u \) is reachable from \( s \), then \( D[u] \) is the length of the shortest simple path (no cycles) from \( s \) to \( u \).

A More Detailed Implementation

• With early stopping
  - and checking for negative cycles